ACEEE Int. J. on Recent Trends in Engineering and Technology, Vol. 10, No. 1, Jan 2014

Model-Driven Development of Web Applications

László Barabás[1] and Barna Kakucs[2]
[1] Evoline SRL, Cluj-Napoca, Romania
Email: barabas.laszlo@evoline.ro
[2] Babeș-Bolyai University, Cluj-Napoca, Romania
Email: kakucs.barna@gmail.com

***Abstract*—Over the last few years Model-Driven Development (MDD) has been regarded as the future of Software Engineering, offering architects the possibility of creating artifacts to illustrate the design of the software solutions, contributing directly to the implementation of the product after performing a series of model transformations on them. The model-to-text transformations are the most important operations from the point of view of the automatic code generation. The automatic generation or the fast prototyping of applications implies an acceleration of the development process and a reduction of time and effort, which could materialize in a noticeable cost reduction. This paper proposes a practical approach for the model-based development of web applications, offering a solution for the layered and platform independent modeling of web applications, as well as for the automatic generation of software solutions realized using the ASP.NET technology.**

***Index Terms*—model-driven development, domain-specific modeling language, model-to-text transformation, platform independent modeling, template-based code generation, web applications.**

I. Introduction

In today's technologically advanced world the internet plays a key role in the lives of individuals. The comfort offered by the internet is quite significant. One can browse for information, perform transactions or purchases from their desk at home. Almost any problem has a solution, regardless of being a web or a mobile application. The well known phrase "There's an app for that!" is a worthy illustration of this fact.

The software development process consists of far more steps than just the implementation. The detailed documentation of the solution plays an equally important role, including the design of the software's architecture, this being quite a costly process. Although the perpetual release of engineering tools and frameworks does contribute to the speedup of the development, it would be extraordinary if there was a possibility for the software engineers to reuse the models illustrating the architecture of the solutions in order to create at least a prototype of the application, thus saving even more development effort and time.

This is the principle used in model-driven engineering, in which case some model transformations enable the automatic generation of documents, and most importantly, source code.

Aiming to accomplish the above mentioned time and effort save, this study proposes a new approach for the model-driven development of web applications, specifically those implemented using the ASP.NET technology. First of all, the paper offers a solution for the platform independent modeling of such software products. Secondly, it provides a tool that, by processing the models created in the documentation phase of the development process, generates a fully functional ASP.NET project that only requires a pre-launch build in order to run, without any further configurations.

The paper is structured into eight sections, as follows: the next section of the document presents some related

DOI: 01.IJRTET.10.1.1009

approaches to the topic covered in this study. Section 3 presents the treated problem generally. Section 4 briefly describes the technologies used for implementing the solution, along with some theoretical considerations as well. Section 5 presents the domain-specific modeling language used for the platform independent modeling. Section 6 illustrates the architecture of the code generation tool, and the steps of the automatic source code generation process. Section 7 creates a brief comparison between the results of the current study and those of a related one. Finally, the last section of the paper discusses the future development possibilities and creates a summary about the advances made during the research.

II. Related Work

To date, OO-HDM [1] and WebML [2] are the two most wide-spread web application modeling approaches.
OO-HDM (Object-Oriented Hypermedia Design Method) is one of the first approaches that support the "Separation of Concerns" principle. It is structured into five steps: requirements analysis, conceptual design, modeling of navigational possibilities, modeling of the abstract user interface, and the implementation.
The WebML approach proposes a visualization language and an appropriate methodology, usable for web applications interacting with data having a complex structure. The methodology proposes the use of five model categories: the structural model, the navigation model, the derivation model, the composition model and the presentation model.
Kateros, Kapitsaki, Tselikas and Venieris observed, that the above mentioned methodologies focused on information modeling rather than the functional design of the applications [3]. They proposed an approach which used class and state transition diagrams, using the elements of a UML profile, for the modeling of web applications. The transformations performed on these models lead to the generation of JSP-specific source code and the appropriate configuration files.
Template-based code generation has been applied in the case of the study carried out by Kroiß [4], using the Java Emitter Templates, an Eclipse instrument. The more actual study presents a new web application modeling approach called the UML-based Web Engineering (UWE) [5]. The UWE methodology was meant to illustrate the entire life-cycle of a web application. The research carried out by Kroiß materialized in an Eclipse extension, called UWE4JSF. The instrument generates JSF-specific source code from the UWE models created in MagicDraw, after a laborious configuration process.
The approach presented in this paper also implements a template-based code generation, as presented in Section V of the document.

III. Problem Statement

As prefaced in the introduction, in recent years web applications have gained significant ground against desktop applications, not only due to the fact that their accessibility is unlimited by space, but thanks to the advances made on the plan of their security as well. Based on recent studies, more and more people are opting to perform their purchases and do their banking online on a daily basis. Thanks to these advances the development of high quality web applications is becoming a priority in the world of Software Engineering (SE). The continuous surfacing of instruments favoring the speedup and quality improvement of web applications supports this hypothesis. Although MDD is being regarded as the future of SE, there are only a handful of high quality instruments supporting the MDD of web applications. Some of these have already been briefly described. However, ASP.NET is not supported by any of the mentioned instruments.
Furthermore, in order to use a MDD approach for web applications one needs a web-domain-specific modeling language. This language should be platform independent in order to support the development of applications implemented using various technologies. After the language constructs have been defined, there is a need for a set of model transformations. The model transformations can and should be platform-specific, because they result in source code generation.
The study proposes a new web modeling approach encapsulating the EAUWE DSML, able to describe the entire lifecycle of a web application, and the EAUWEGenerator Enterprise Architect Add-in containing the appropriate model transformations for source code generation. The aimed implementation technology is the ASP.NET. The approach is based on the UWE approach, a well-documented methodology developed at the Ludwig-Maximilians University from Munich [5].
The next section presents the technologies used for carrying out the study for implementing the code generator and those used by the web applications generated automatically by the code generation instrument.

IV. Used Technologies

The approach presented in this document describes the entire cycle of the MDD, including the modeling phase and the model-to-text transformations. A modeling tool was used in the process of studying the MDD.
The software industry promotes the use of software frameworks in the development process. In order to fulfill this expectation, different tools and frameworks were used in case of the generated projects, in order for them to be considered cutting edge applications.
These technologies and theoretical considerations are briefly presented in the following subsections.

A. Domain-Specific Modeling Languages and the Unified Modeling Language

The Unified Modeling Language (UML) [6] is a standard modeling language, being probably the most wide-spread modeling language. It enables not only the modeling of the architecture, the behavior and the structure of an application, but the contained business processes as well. Whenever the UML models are incapable of offering enough specification power to detail a given functionality, custom UML profiles can be used. Abouzahra [7] stated that the UML profiles were created with the specific goal of formalizing and supporting the development of scientific domain-specific applications.
The phrase "domain-specific modeling language" (DSML) represents a specification language in terms of software engineering. It is used to describe a problem belonging to a particular scientific domain.
This kind of languages can be considered the opposite of the UML specification standard, which can only be used to describe and model a general solution. There are two methods for defining a DSML: a metalanguage or a UML profile for defining the stereotypes. Tolvanen and Kelly in ref. [8] related about the necessity of using multiple models/diagrams for specifying a given problem and describe a handful of integration possibilities between models and DSMLs, in order to achieve a greater specification power.
This study used the second modality of specifying DSMLs, which is to be described in Section V.

B. The ASP.NET Technology

As a part of the .NET Framework [9], the ASP.NET [10] technology is quite wide-spread. It is a server-side technology, which applies the principles of Object-Oriented Programming to the elements of web applications, offering a modality for creating dynamic web pages and web services.
In an ASP.NET page every web and server control is treated as an object, and is run on the server.
IDEs like the Microsoft Visual Studio and the Visual Web Developer promote the development of these kinds of applications, and they even provide a visual developing user interface for the less experienced.

C. The ADO.NET Entity Framework

The Entity Framework (EF) is one of the components of ADO.NET. The EF is an Object/Relational Mapping (ORM) framework, which enables the data mapping between the conceptual model and the developed product using an Entity Data Model. The innovation of the EF resides in the fact that it provides a strongly typed interface, which enables the object-oriented querying and manipulation of data of relational databases.
The next section presents an approach on the platform-independent modeling of web applications.

V. Specifying the EAUWE DSML Using A UML Profile

In recent years new approaches have surfaced on the topic of the modeling of web applications. The UWE [5] modeling methodology is considered the foundation of the modeling possibilities offered in this study.
As the definition for the DSML a UML profile has been defined, the UWE UML profile was recreated with the appropriate stereotypes for usage in Enterprise Architect (EA), the modeling instrument of choice. The EAUWE DSML offers a modeling possibility to create a layered architecture for the software solution.
In Software Engineering the content model encapsulates static structures that reflect the structure of the modeled system. The view is represented in diagram form as a "conventional" class diagram. Based on this property, classes serve as foundation for this model. Due to its outstanding expressivity, neither the creators of the UWE approach, nor those of the current study have contributed to the view with groundbreaking innovations. Nevertheless, some stereotypes have been relegated in comparison to the original approach due to some technical incompatibilities. For instance, the "evaluatedProperty" and "evaluatedOperation" stereotypes symbolized elements whose value expressions were given using the Object-Graph Notation Language (OGNL). The OGNL is specialized in Java, so it could not have been used in case of .NET source code. In order to balance out the absence of the omitted keywords, the EA offered a possibility to specify initial C# code in case of properties or methods.

The elements of the User View correspond to those of the Content View, although each user stereotype is prefaced with the term “user”. If the similarity is so evident, why is there a need for the User model?
While the Content elements have an optimized representation in the relational database the generated application will interact with, the User model elements have no such representation. Furthermore, their instantiation and usage in restricted to only a single Session of the software solution. This fact motivates the “Session-Object” property of the User model elements.
The EAUWE Use Case View, alongside the standard Use Case model stereotype family, contains new elements as well. The new stereotypes introduced include, for instance, the “navigation”, the “process” and the “webUseCase” elements. For the complete specification of the stereotypes, please refer to [11].
The elements of the Navigation view serve to illustrate the totality of navigation paths in a software solution. Therefore, it contains no restrictions depending on the rights of the active user, nor does it contain constraints for the navigation templates of different activities. As specified in [11], the UWE Navigation view contains a handful of stereotypes. The majority of these stereotypes have been readapted in order to be usable in EA, some others have been omitted, because they did not play a relevant role in this study.
The Process View of the EAUWE Profile contains the Process Flow and Process Structure models, just like in the case of the UWE approach. The Process Flow model is a readapted Activity model, the unique name represents the model’s appurtenance to the EAUWE approach. On the other hand, the Process Structure model is quite specific. The activities represented in it can be categorized into four types: systemAction, userAction, displayAction and navigationAction, their names having been chosen quite intuitively.
Contrary to the Navigation model, the Process model is more restrictive, and, though it includes the representation of many use cases, the Process Flow diagrams always illustrate the progress template of a single use case. This provides the “more restricted” nature of the Process View.
The elements of the Presentation View enable the specification of the User Interface (UI) of the modeled web application in an object-oriented fashion. On the one hand, the EAUWE Presentation View does not contain all of the stereotypes from the UWE approach; on the other hand, it contains new elements, and some of the readapted elements were given new meanings. The significance of the elements is readapted in order to conform to the targeted .NET technology. The original connotation of the elements was described in [11].
The UML profile briefly presented in this section describes the EAUWE domain-specific modeling language. The following section describes the EAUWEGenerator EA Add-in, along with the steps of the template-based code generation performed by the above mentioned tool.

VI. Automatic Source Code Generation for EAUWE Models

This section contains the essence of the paper, describing the final product that extends the usage possibilities of the EAUWE models to implement web applications using the ASP.NET technology. Being an EA Add-in, the EAUWEGenerator uses EAUWE models as input and produces ASP.NET-specific source code.
The generated source code corresponds to the content, user, presentation views of the model. The persistence layer is automatically generated also. The following subsections present the processing methodology, and the template-based code generation approach adapted in the EAUWEGenerator.

A. Processing the EAUWE Models

During the modeling process, the models of a project created using the Enterprise Architect instrument are stored in a compressed file having the .eap extension. The proprietary nature of the EAP file format makes its manual processing difficult. However, EA offers the possibility to create different extensions, for instance, for the .NET technology, a library by the name of “Interop.EA”. The library contains the complete API offered by EA and it facilitates the processing of the models created using the EAUWE profile.
It also offers an OO interface that, like the modeling process itself, is platform independent. Consequently, the elements of the interface are not directly usable in the code generation process that results in .NET-specific source code. For solving this problem, an equivalent platform-specific class or enumeration is defined inside the EAUWEGenerator instrument for every API element/entity, which overtakes the model element’s content. Based on these considerations, the architecture of the EAUWEGenerator has been defined so that it contains a tree structure for representing the elements, presented in Figure 1.
As it can be seen in Fig. 1, the unconventional tree structure incorporates the essence of the research, because, after parsing it, the code generation is a simple formality. The root of the tree structure is the element called EAProject, which offers the possibility of parsing the entire model. The first level of the tree contains the different namespaces obtained from the model categories from the modeling project.

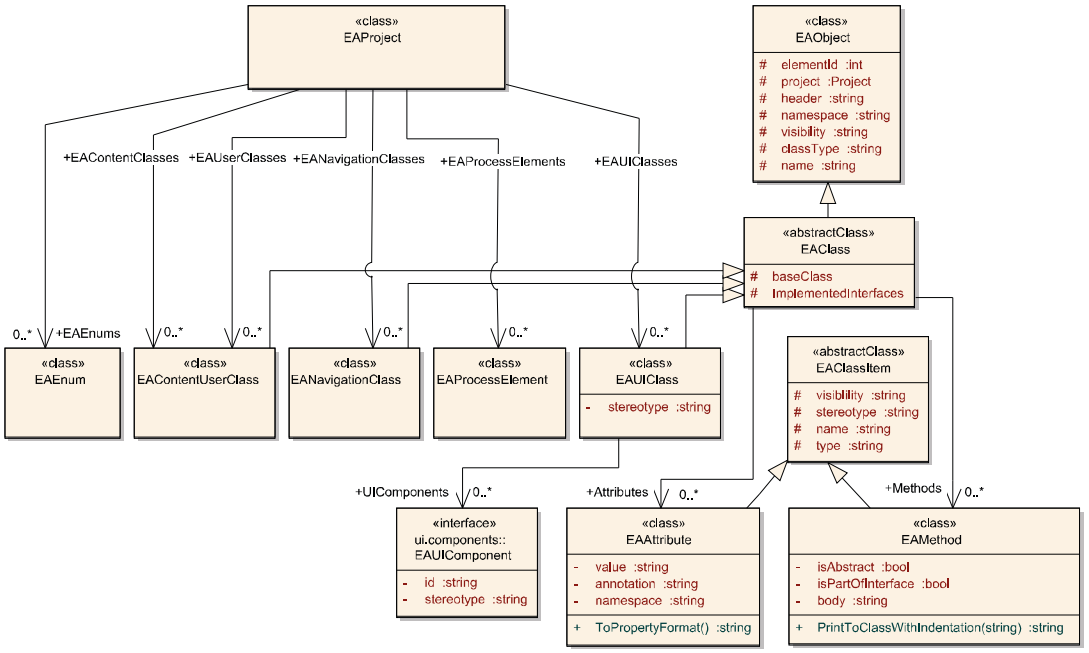

Figure 1. The simplified and partial architecture of the tree structure from the EAUWEGenerator

The next level of the tree contains the different classes and structures belonging to the defined namespaces.

The third level of the tree consists of the EAUIComponent interface, which symbolizes a general user interface (UI) element. Each model element of the Presentation View implements the mentioned interface. Evidently, each one of these classes will have a source code equivalent after the code generation process.

The aggregation of the elements implementing the EAUIComponent interface and of the elements inheriting the abstract EAClassItem can be considered the leaves of the tree structure. The inheriting members are the EAAttribute class, representing the .NET-specific Properties of classes, and the EAMethod class, which symbolizes the functions or methods of the modeled elements.

The decision of building the tree structure was motivated by the theoretical model of the abstract syntax tree (AST) and by its significant role in the source code generation process. However, the tree built during this study is very different from the AST, mainly because of the significant difference in the abstraction level.

The left side of Fig. 2 represents a trivial partial Content model: practically it illustrates two classes, the Garden and the Flower classes, along with their properties and functions. The Garden consists of three properties and a function, the Flower three properties. This means a total of nine model elements that will have source code equivalents grouped into two files, one class per file. The right side of Fig. 2 illustrates the simplified structure of the tree built during model parsing from the given Content model.

The next subsection treats the actual source code generation process during which the tree structure, which has been built during the EAUWE model parsing, is traversed in a depth-first fashion.

B. The Actual Source Code Generation Process

In the previous subsection it was mentioned that after processing the EAUWE models from a modeling project, source code is automatically generated by iterating through the built tree structure. This section presents the methodology of iterating through the tree, and of the code generation.

As a first step of the code generation some configuration data has to be specified. This data includes the name of the SQL server and the database the generated application will interact with, along with the path the web application will be automatically generated to.

The depth-first traversal of the built tree structure constitutes the second step of the source code generation process. This way, all classes from all namespaces are traversed, along with the elements' appropriate attributes and methods. Each component of the architecture described in the previous section has an equivalent template for source code generation. These components determine the "template-based" nature of the code generation process performed by the EAUWEGenerator instrument.

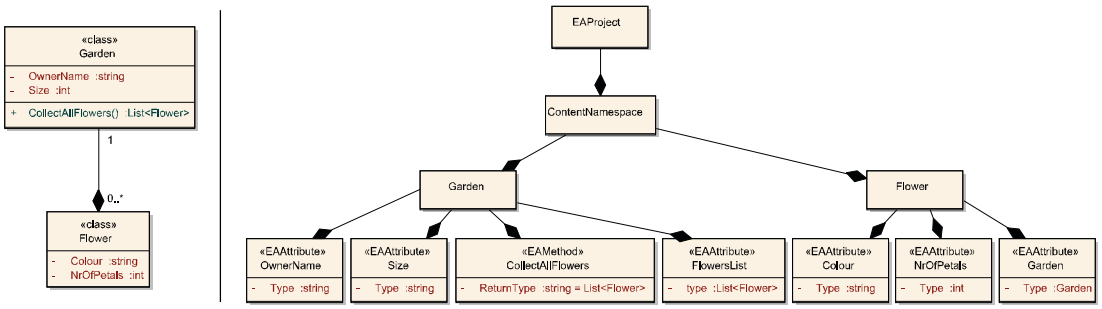

Figure 2. A simple partial Content model (left) and its parsed simplified tree-structure equivalent (right)

The template-based source code generation is quite unconventional. While in the classic approaches the templates are static, in this case the dynamicity is dominant. The classic templates are static text files that are read, parsed, completed, and only then written to files with appropriate names. Contrary to the usual approach, in the case of the EAUWEGenerator a parsed class is transformed into a text file with the appropriate name, after traversing the tree in a depth-first fashion. The assembly of the completed template is performed using a StringBuilder, rather than a String from the usual approach.

Consequently, in the case of the EAUWEGenerator not only the templates are more dynamic, but the formatting instrument offers more advantages, as well, compared to the classical approach.

C. The Source Code Generated for the Different Layers of the Application

This section of the study creates a link between the model elements and their source code equivalent.

Firstly, the Use Case model is entirely theoretical and is not applicable for source code generation.

Secondly, based on the definition, the Navigation model's elements are linked to those of the Content model; that is why the navigation classes do not have any representation in the generated source code on their own.

Thirdly, the Process model is used to represent different business processes, and their steps, and can be detailed using Process Flow diagrams. The Process Flow diagrams are an extension of Activity diagrams, and, although they contain business logic elements, they did not play a role in the code generation process performed by the EAUWEGenerator.

As clarified during the description phase of the Content View of the EAUWE Profile, it contains classes, interfaces, abstract classes and enumerations. Their source code equivalent is quite evident. Each generated text file contains a single class, interface or structure, with the appropriate name. It also contains all the appropriate "using" directives, the namespace name, the access modifier, and the actual content of the element. Furthermore, in the case of the elements other than enumerations, the .NET-specific properties of the source code affirm themselves even more, because the elements do not contain classic fields with the appropriate getter and setter methods, instead, they possess Properties, encapsulating the two methods.

In the case of functions and methods, as specified in Section V, initial source code can be specified during the modeling process. This is significant in case of functions, because it can lead to compilation errors.

In case of certain properties database key-related annotations are specified automatically. The motivation of this fact is to be presented in the following Persistence-related paragraph.

Although the element types from the User model coincide with those of the Content model, the main difference between them is that the User model elements are used inside a single session of the application, so they do not have any representation in the relational database the web application interacts with. Consequently, these elements are not added to the database context class, nor do they contain any key-related annotations, either.

The Content model elements can contain some annotations that are meant to facilitate the realization of the Persistence layer of the web application. The ADO.NET EF has been used for creating the Persistence layer. Using the framework, a strongly-typed interface is generated, which enables the manipulation of data from relational databases. In addition, there is a need for a database context class for data manipulation, based on the Code First approach of the EF.

The EF is quite helpful in creating the optimized equivalent relational databases for the source code models. However, in case of 1-to-1 relationships between model elements the instrument cannot decide which is the Principal End of Association in the relationship. This issue can be solved using the above mentioned annotations. This way the database generation process, the mappings and the data manipulation can be

carried out without any further problems.
The EF reads the connection string from the Web.config file, whose content is generated automatically also, using the name of the SQL server and the database the web application will interact with.
The core element of the Presentation model is the one by the name of PresentationPage, which models an entire ASP.NET page. This way its content is separated into three source files: one with the .aspx extension, an HTML-like file, one with .aspx.cs extension, file that contains the event handler methods of the encapsulated web and server controls, and, finally, one with the .aspx.designer.cs extension, containing the definitions of all server and web controls from the web page.
Each HTML element is either treated as a web or server control, possessing the "runat="server"" attribute. This way the user action event handling is carried out in an object-oriented manner. Some of the UI model elements are described in the following paragraph.
First of all, the element having the anchor stereotype represents an HTML anchor. The button symbolizes an ASP:Button, having an implicit event handler method. The fileUpload stereotype marks an input with the type of file. The list marks an HTML list. The presentationGroup marks an HTML div. The selection represents an HTML select, the text stereotype points to a simple text fragment. The textInput stereotype can symbolize a textbox or a textarea, depending on its "multiline" tagged value.
The obtained text files are structured under a Project file, specific for the Visual Studio 2010 IDE. The project file is the result of automatic generation too, using the functionalities of the Visual Studio Project Templates. After the "classic" .NET libraries are added to the project file, along with those that are necessary for the EF, the AssemblyInfo, Web.config and other configuration files, the generated source files are added to the project to their respective namespaces.
This way, the layered architecture of the modeled web application is visible on the source code level as well.

D. Code Generation Results Summary

After generation, the resulted ASP.NET web applications have a layered architecture, which can include a content layer, containing the conceptual model of the web application, a persistence layer, using which the manipulation of data from relational databases is carried out, a user layer, containing elements of session-object type, and a presentation layer, namely, the user interface of the application.
Consequently, after the code generation process is carried out by the EAUWEGenerator, the result is a complete web application, that, after compilation, can be run without any further modifications.

VII. Comparing the Code Generation Instruments from the UWE and EAUWE Approaches

This section contains a brief case study creating a comparison between the UWE4JSF, the code generation instrument of the UWE approach, and the EAUWEGenerator, created during the current study.
As Table I illustrates, the two tools seem to be very similar, but in some key aspects they are quite different. For instance, in case of the Presentation Layer of an application generated using the UWE4JSF some inconsistencies were noticed and the Persistence Layer presented some shortcomings as well.
The EAUWEGenerator claims to generate runtime error-free source code. However, errors might occur in case of modeling errors that were undetectable during validation. Such error occurs in the case of a request to an inexistent page, which is concluded in a error with the 404 error code. The generated Process Layer is the

TABLE I. COMPARING THE FUNCTIONALITIES OF THE UWE4JSF AND EAUWEGENERATOR INSTRUMENTS

Characteristics	UWE4JSF	EAUWEGenerator
Implementation technology of the generated source code	JSF	ASP.NET
Content Layer Generated, User Layer Generated, Presentation Layer Generated, Persistence Layer Generated	Yes	Yes
Process Layer Generated	Yes	No
Framework Used for the Generation of the Persistence Layer	Hibernate	Entity Framework
Post-generation No Necessary Manual Configuration Before Run	No	Yes
Application Generated without Compilation Errors	Yes	Yes
Application Generated without Runtime Errors	No	Yes
Accessibility of the Instrument	External Eclipse Plugin	Directly from EA

only one that is absent from the EAUWEGenerator compared to its counterpart.
The stability of the instrument compensates for this deficit: the source code issued by the EAUWEGenerator is always stable and functional.
In real-life software development processes the use of frameworks is a necessity. Both of the compared instruments excel at this aspect, using the Hibernate and EF for data manipulation.
Probably the most important aspects of an instrument are its usability, accessibility and user-friendliness. While in the case of the UWE4JSF one has to use Eclipse Galileo, a deprecated version of the IDE, along with a series of manually installable packages, and import the models, the user can generate code directly from EA with the EAUWEGenerator, due to the fact that the it is an EA add-in. This way, on the plan of the user-friendliness and usability the instrument realized in this study eclipses its counterpart.
Based on the aspects presented in this section one can conclude that the EAUWEGenerator is veritable competition for the tool from the UWE approach. Furthermore, due to the fact that it generates ASP.NET code, it is unique on the software market, which can confer an even higher significance to the instrument.

VIII. Conclusions and Future Work

In today's technologically advanced world the documentation phase of the development of software solutions plays a key role, followed by the actual implementation of the product. This way it would be quite beneficial and time- and effort-saving if, after the documentation phase, at least a frame of the developed web application was already generated automatically.
The study proposed a solution to this problem, by extending the UWE UML profile and creating a source code generation instrument, the EAUWEGenerator Enterprise Architect add-in. The created tool performs a model-to-text transformation on the models created using the EAUWE profile elements and generates source code files respecting the standards of the ASP.NET technology. Given the fact that the instrument generates the content, user, persistence and presentation layers of the modeled web application, the development process can be significantly accelerated, which is materialized in a noticeable cost reduction.
As for future plans, the EAUWE DSML is proposed to be enriched with new elements, mainly the Presentation view of the UML profile, in order for the modelers to be able to use a broader spectrum of HTML elements for modeling the UI of the web applications. This will also imply some modifications in the code generation instrument.
Secondly, a whole new version of EAUWEGenerator is proposed to be developed in order to generate source code specific to the ASP.NET MVC4 technology, a more recent approach proposed by Microsoft.
On the software market there is a lack of high quality instruments for the model-driven engineering of web applications. Consequently, based to the complexity and stability of the instrument, the EAUWEGenerator product can be considered a strong alternative for the generation/fast prototyping of ASP.NET applications.

References

[1] G. Rossi and D. Schwabe, "Developing Hypermedia Applications using OOHDM", Proceedings of the ninth ACM Conference on Hypertext (Hypertext '98), 1998
[2] S. Ceri, P. Fraternali, A. Bongio and P. Bongio, "Web Modeling Language (WebML): a modeling language for designing Web sites", Computer Networks, vol. 33, ed. 1–6, pp. 137–157, June 2000
[3] D. A. Kateros, G. M. Kapitsaki, N. D. Tselikas and I. S. Venieris, "A Methodology for Model-Driven Web Application Composition", SCC '08. IEEE International Conference onServices Computing, vol. 2, pp. 489-492, 2008
[4] Ch. Kroiß, „Modellbasierte Generierung von Web-Anwendungen mit UWE", http://uwe.pst.ifi.lmu.de/publications/christian_kroiss_Ausarbeitung_DA_final.pdf, unpublished (June 11,2013)
[5] N. Koch, "Model-Driven Web Engineering: UWE Approach", http://uwe.pst.ifi.lmu.de/publications/MDWE-UWE_URJC_280508.pdf, unpublished (June 11,2013)
[6] ***, "Unified Modeling Language", http://www.uml-diagrams.org/profile-diagrams.html (June 11,2013)
[7] A. Abouzahra, "Bridging UML Profiles and Domain Specific Languages", unpublished, 2005
[8] J.-P. Tolvanen and S. Kelly, "Integrating Models with Domain-Specific Modeling Languages", Proceedings of the 10th Workshop on Domain-Specific Modeling, Article No. 10, 2010
[9] ***, .NET Framework, http://msdn.microsoft.com/en-us/library/vstudio/w0x726c2%28v=vs.100%29.aspx, (June 11,2013)
[10] ***, ASP.NET, http://www.asp.net/ (June 11,2013)
[11] ***, "UWE Metamodel and Profile – User Guide and Reference", http://uwe.pst.ifi.lmu.de/download/UWE-Metamodel-Reference.pdf, 2008 (June 11,2013)